

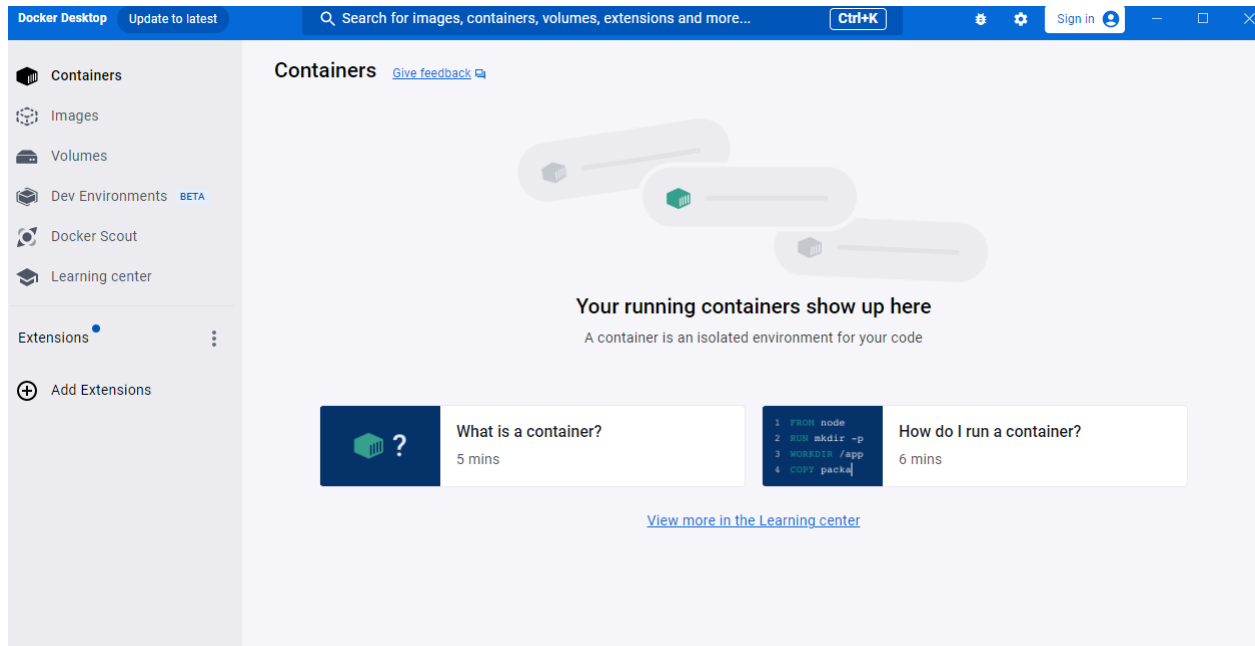
Adv DevOps Exp 06

Aim: To create docker image using terraform

Step 1: Check the functionality of docker desktop i.e Check if Docker is installed properly on your system and that Docker Desktop is opening properly (virtualization settings are needed to be enabled in our system bios)

```
C:\Users\INFT505-16>docker -v
Docker version 24.0.6, build ed223bc

C:\Users\INFT505-16>|
```



Step 2: Now, create a folder named 'Terraform Scripts' in which we save our different types of scripts which will be further used in this experiment.

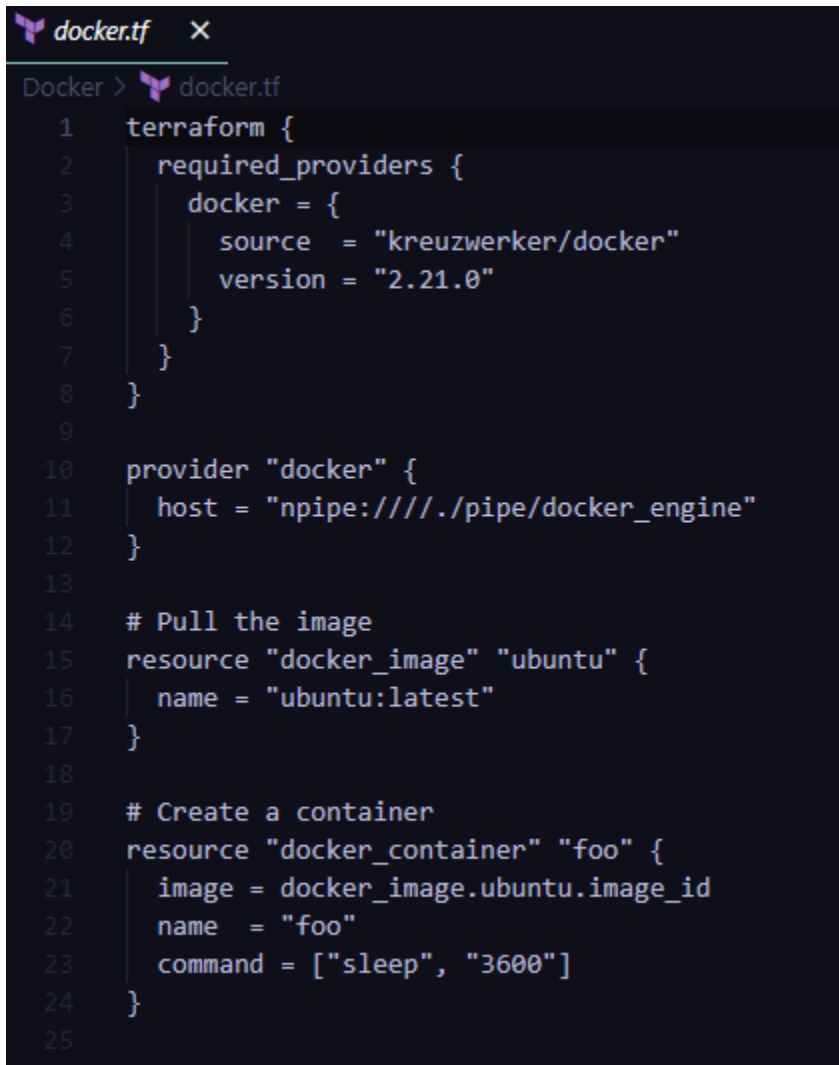
Step 2: Firstly create a new folder named 'Docker' in the 'TerraformScripts' folder. Then create a new docker.tf file using Atom editor and write the following contents into it to create a Ubuntu Linux container.

Script:

terraform

```
{ required_providers
{docker = {
source = "kreuzwerker/docker"
version = "2.21.0"
```

```
}  
}  
}  
provider "docker" {  
  host = "npipe:////./pipe/docker_engine"  
}  
# Pulls the image  
resource "docker_image" "ubuntu"  
{name = "ubuntu:latest"  
}  
# Create a container  
resource "docker_container" "foo"  
{ image =  
  docker_image.ubuntu.image_idname =  
  "foo"  
}
```



```
docker.tf X  
Docker > docker.tf  
1 terraform {  
2   required_providers {  
3     docker = {  
4       source = "kreuzwerker/docker"  
5       version = "2.21.0"  
6     }  
7   }  
8 }  
9  
10 provider "docker" {  
11   host = "npipe:////./pipe/docker_engine"  
12 }  
13  
14 # Pull the image  
15 resource "docker_image" "ubuntu" {  
16   name = "ubuntu:latest"  
17 }  
18  
19 # Create a container  
20 resource "docker_container" "foo" {  
21   image = docker_image.ubuntu.image_id  
22   name = "foo"  
23   command = ["sleep", "3600"]  
24 }  
25
```

Step 3: Run terraform init command

Run the command **terraform init** in your terminal. This command initializes the Terraform configuration, downloads the necessary provider plugins specified in the **docker.tf** file, and prepares the working directory for further commands. This step is crucial to ensure that Terraform can interact with Docker.

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of kreuzwerker/docker from the dependency lock file
- Using previously-installed kreuzwerker/docker v2.21.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>
```

Step 4: Run terraform plan command:

Next, execute the command **terraform plan** to create an execution plan. This command will show what actions Terraform will take based on your configuration. It ensures that the planned changes match your expectations before actual implementation.

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
+ create

Terraform will perform the following actions:

# docker_container.foo will be created
+ resource "docker_container" "foo" {
+   attach      = false
+   bridge      = (known after apply)
+   command     = [
+     "sleep",
+     "3600",
+   ]
+   container_logs = (known after apply)
+   entrypoint    = (known after apply)
+   env          = (known after apply)
+   exit_code     = (known after apply)
+   gateway      = (known after apply)
+   hostname     = (known after apply)
+   id           = (known after apply)
+   image        = (known after apply)
+   init         = (known after apply)
+   ip_address    = (known after apply)
+   ip_prefix_length = (known after apply)
+   ipc_mode     = (known after apply)
+   log_driver    = (known after apply)
+   logs         = false
+   must_run     = true
+   name         = "foo"
+   network_data  = (known after apply)
+   read_only    = false
+   remove_volumes = true
}
```

```

+ rm                = false
+ runtime           = (known after apply)
+ security_opts     = (known after apply)
+ shm_size          = (known after apply)
+ start             = true
+ stdin_open        = false
+ stop_signal       = (known after apply)
+ stop_timeout      = (known after apply)
+ tty               = false

+ healthcheck (known after apply)

+ labels (known after apply)
}

# docker_image.ubuntu will be created
+ resource "docker_image" "ubuntu" {
  + id            = (known after apply)
  + image_id      = (known after apply)
  + latest        = (known after apply)
  + name          = "ubuntu:latest"
  + output        = (known after apply)
  + repo_digest   = (known after apply)
}

Plan: 2 to add, 0 to change, 0 to destroy.

```

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.

C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>

Step 5: Execute terraform apply command:

Run **terraform apply** to apply the configuration and create the specified resources. During this step, Terraform will prompt you to confirm the action by typing yes. Upon confirmation, Terraform will pull the Ubuntu image and create the container, reflecting successful deployment.

```

C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with:
+ create

Terraform will perform the following actions:

# docker_container.foo will be created
+ resource "docker_container" "foo" {
  + attach          = false
  + bridge          = (known after apply)
  + command         = [
    + "sleep",
    + "3600",
  ]
  + container_logs  = (known after apply)
  + entrypoint      = (known after apply)
  + env             = (known after apply)
  + exit_code       = (known after apply)
  + gateway         = (known after apply)
  + hostname        = (known after apply)
  + id              = (known after apply)
  + image           = (known after apply)
  + init            = (known after apply)
  + ip_address      = (known after apply)
  + ip_prefix_length = (known after apply)
  + ipc_mode        = (known after apply)
  + log_driver      = (known after apply)
  + logs            = false
  + must_run        = true
  + name            = "foo"
  + network_data    = (known after apply)
  + read_only       = false
  + remove_volumes = true
}

```

We are supposed to put yes to proceed ahead

```
+ rm                = false
+ runtime           = (known after apply)
+ security_opts     = (known after apply)
+ shm_size          = (known after apply)
+ start             = true
+ stdin_open        = false
+ stop_signal        = (known after apply)
+ stop_timeout      = (known after apply)
+ tty               = false

+ healthcheck (known after apply)
+ labels (known after apply)
}

# docker_image.ubuntu will be created
+ resource "docker_image" "ubuntu" {
  + id          = (known after apply)
  + image_id    = (known after apply)
  + latest      = (known after apply)
  + name        = "ubuntu:latest"
  + output      = (known after apply)
  + repo_digest = (known after apply)
}
```

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

After executing terraform apply:

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

docker_image.ubuntu: Creating...
docker_image.ubuntu: Still creating... [10s elapsed]
docker_image.ubuntu: Creation complete after 12s [id=sha256:b1e9cef3f2977f8bdd19eb9ae04f83b315f80fe4f5c5651fedf41482c12432f7ubuntu:latest]
docker_container.foo: Creating...
docker_container.foo: Creation complete after 1s [id=684alde22619abba495d0c2886fc9d8f681302574f3e1ca16c5297470b4e88b4]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>
```

Terraform images after apply step:

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
ubuntu          latest      b1e9cef3f297  3 weeks ago   78.1MB
sonarqube       latest      3183d6818c6e  11 months ago 716MB

C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>|
```

Step 6: Run terraform destroy command

To remove the resources created during the experiment, run the command **terraform destroy**. This command will prompt for confirmation before proceeding to delete the container and any other associated resources. It's essential for maintaining a clean working environment and avoiding unnecessary resource consumption.

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:b1e9cef3f2977f8bdd19eb9ae04f83b315f80fe4f5c5651fedf41482c12432f7ubuntu:latest]
docker_container.foo: Refreshing state... [id=684a1de22619abba495d0c2886fc9d8f681302574f3e1ca16c5297470b4e88b4]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- destroy

Terraform will perform the following actions:

# docker_container.foo will be destroyed
- resource "docker_container" "foo" {
  - attach      = false -> null
  - command     = [
    - "sleep",
    - "3600",
  ] -> null
  - cpu_shares  = 0 -> null
  - dns         = [] -> null
  - dns_opts    = [] -> null
  - dns_search  = [] -> null
  - entrypoint  = [] -> null
  - env         = [] -> null
  - gateway     = "172.17.0.1" -> null
  - group_add   = [] -> null
  - hostname    = "684a1de22619" -> null
  - id          = "684a1de22619abba495d0c2886fc9d8f681302574f3e1ca16c5297470b4e88b4" -> null
  - image       = "sha256:b1e9cef3f2977f8bdd19eb9ae04f83b315f80fe4f5c5651fedf41482c12432f7" -> null
  - init        = false -> null
  - ip_address  = "172.17.0.2" -> null
  - ip_prefix_length = 16 -> null
  - ipc_mode    = "private" -> null
  - links       = [] -> null
  - log_driver  = "json-file" -> null
  - log_opts    = {} -> null
```

After executing terraform destroy command:

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
sonarqube       latest      3183d6818c6e  11 months ago 716MB

C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>|
```

Conclusion: In this experiment, we effectively utilized Terraform to manage Docker resources by creating an organized folder structure and executing a series of commands to deploy an Ubuntu container. By following the steps from initialization to resource destruction, we gained hands-on experience in infrastructure as code (IaC) principles, enhancing our understanding of managing containers with Terraform and Docker. This practical approach not only solidified our knowledge but also emphasized the importance of automated resource management in modern DevOps practices.